Oblivious Routing Schemes in Extended Generalized Fat Tree Networks

German Rodriguez[1], Cyriel Minkenberg[2], Ramon Beivide[3], Ronald P. Luijten[2], Jesus Labarta[1], Mateo Valero[1]

[1] Barcelona Supercomputing Center, Nexus II, C/ Jordi Girona, 29, 08034 Barcelona, Spain
[2] IBM Research, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland
[3] Universidad de Cantabria, Avenida de los Castros s/n, 39005 Santander, Spain

***Abstract*—A family of oblivious routing schemes for Fat Trees and their slimmed versions is presented in this work. First, two popular oblivious routing algorithms, which we refer to as S-*mod*-k and D-*mod*-k, are analyzed in detail. S-*mod*-k is the default routing algorithm given as an example in the first works formally describing Fat Tree networks. D-*mod*-k has been independently proposed and investigated by several authors, who conclude in their evaluations that it achieves better performance than a *random* or *adaptive* routing approach. First, we identify the reasons why these algorithms perform well. Using this insight we extend these algorithms, originally intended for full bisection networks, to slimmed networks. Based on the lessons learned we propose a new generalized family of algorithms that provides a better oblivious solution than the existing ones for this class of networks. Moreover, this family extends the previous work from k-ary n-trees to the more general class of extended generalized fat trees.**

I. Introduction

Fat Tree networks have been present in many High Performance Computing (HPC) machines since they were first introduced in a popular HPC system, the Thinking Machines CM-5 in 1991. Deadlock freedom and availability of multiple paths, along with the capability of emulating other well-known networks with a theoretically achievable polylogarithmic slowdown [1] popularized their use. However, despite the huge corpus of theoretical results, studies of their suitability and cost for HPC traffic using real application patterns are few. Recent works [2], [3], [4] arrive at the conclusion that the network is overdesigned and underutilized.

Among many possible realizations of Fat Tree networks, k-ary n-trees are the most widely spread. k-ary n-trees sacrifice connectivity to reduce complexity [5] and provide a full-bisection network using switches all having the same number of ports. The traded connectivity increases the impact of network contention.

Network contention can be mitigitated by an appropriate routing scheme. To this end, many proposals exist, the majority of which are oblivious to the global communication pattern that is taking place. Many adaptive algorithms that take local decisions have also been proposed; however, some works have shown that these are not always better than oblivious algorithms [6]. A very simple oblivious algorithm (D-*mod*-k) has been proved to be very good for a wide collection of patterns [7], [6], [8], [9].

In this work we analyze three static oblivious algorithms (which we will call *Random*, S-*mod*-k and D-*mod*-k) in detail. We will demonstrate that S-*mod*-k, proposed as a default "self-routing" scheme on several works on fat trees [10], [11], and D-*mod*-k perform similarly well, and we will argue that they should lead to similar performance in most cases. We will show that both algorithms concentrate endpoint contention and perform better than *Random*. When extending these algorithms from k-ary n-trees to slimmed trees (which have less connectivity in the upper levels) we discover that another important aspect to gain performance is to distribute endpoint contention evenly across the roots[1] of the tree. Finally, we identify patterns that perform poorly for both S-*mod*-k and D-*mod*-k, because of the regularity of the pattern and the regularity of the assignments of routes to roots of the tree (in this case, *Random* does better). We combine the ideas of concentrating endpoint contention, even distribution of endpoint contention across roots, and randomization to break regularity into a proposal for a new class of oblivious routing algorithms.

The rest of the paper is structured as follows: we will start with a brief description of extended generalized fat tree (XGFT) topologies (Sec. II), communication patterns (Sec. III) and a discussion on contention (Sec. IV). Before delving into the experimental and combinatorial analysis of well-known oblivious routings (Sec. VII) that motivate our proposal (Sec. VIII) we will first describe our evaluation and experimental framework (Sec. VI) as a part of the analysis is based on simulation results obtained to compare several routing schemes. Finally, we will conclude in Sec. X.

II. Extended Generalized Fat Trees

Many current supercomputers employ k-ary n-tree networks [12]. They are a popular parametric family of indirect multi-tree networks. A k-ary n-tree has $N = k^n$ leaf nodes used as processing nodes and $n \cdot k^{n-1}$ inner nodes ($2k$-port switches). These full bisection bandwidth networks exhibit path redundancy and the property of being rearrangeable [13]. This means that any *scheduled* permutation of sources over destinations can be routed without blocking, i.e., no messages contend for the same network port. Each specific permutation needs an appropriate set of routes.

[1] More accurately: Nearest Common Ancestor's (NCAs).

978-1-4244-5012-1/09/$25.00 ©2009 IEEE

Several recent works have identified a potential over-provisioning of bandwidth of k-ary n-trees [2], [3], [14]. Consequently, the use of "slimmed" k-ary n-trees has been considered. Slimmed k-ary n-tree topologies have less than $n \cdot k^{n-1}$ switches, losing both the full bisection bandwidth and the rearrangeable non-blocking properties.

Formally, k-ary n-trees and their slimmed versions belong to the family of XGFTs [10]. This family includes many popular Multi-stage Interconnection Networks (MIN), such as m-ary complete trees, k-ary n-trees [12], fat trees as described in [1], and *slimmed* k-ary n-trees. An XGFT$(h; m_1, ..., m_i, ..., m_h; w_1, ..., w_i, ..., w_h)$ of height h has $N = \prod_{i=1}^{h} m_i$ leaf processors, with the inner nodes serving only as routers. Each non-leaf node in level i has m_i child nodes, and each non-root has w_{i+1} parent nodes [10]. An XGFT of height h has $h+1$ levels. Leaf nodes are at level $l = 0$. XGFTs are constructed recursively, each sub-tree at level l having parents numbered from 0 to $(w_{l+1} - 1)$. See Fig. 1 for some examples.

A k-ary n-tree is an XGFT$(n; \overbrace{k, ..., k}^{n}; 1, \overbrace{k, ..., k}^{n-1})$, where $h = n$, $w_1 = 1$, $m_1 = k$, and $m_i = k, w_i = k$, $\forall i$ with $2 \leq i \leq n$.

A slimmed k-ary n-tree is precisely defined by the vectors w_i and m_i when $\exists i | (w_i < k)$ with $2 \leq i \leq n$. *Slimmed* trees are blocking networks. Figure 1 shows several slimmed trees.

In both cases, the number of inner switches I can be computed as

$$I = \sum_{i=1}^{h} \left(\prod_{j=i+1}^{h} m_j \cdot \prod_{j=1}^{i} w_j \right). \quad (1)$$

At a particular level l, there are $\prod_{j=l+1}^{h} m_j \cdot \prod_{j=1}^{l} w_j$ nodes. A vector of size h can be defined to label each of the nodes (leaf nodes ($l = 0$) are the hosts), and similar tuples of length $h+1$ can be defined to identify both the links up and down to each level (where the links up from level l take identical labels to the ones down from level $l+1$).

In a k-ary n-tree, a node label can be expressed as an n-digit base-k number. In an XGFT, on the other hand, the base of each digit can be different. Table I shows how we label the nodes at each level. The processing nodes are labelled using m_h through m_1 as the base numbers. For each level up the tree, the corresponding base m_i is replaced by w_i; as a result, the top level switches are labelled using w_h through w_1 as the base numbers.

The number of links at level i simply equals the number of nodes N_i times m_i (down) or w_{i+1} (up). As indicated in I the number of links up from level i equals the number of links down from level $i+1$.

III. Communication Patterns

A communication pattern C can be described by a connectivity matrix $M(N \times N)$. The connectivity matrix M of C records its set of connections with elements $m_{ij} \neq 0$ *iff* the connection $(i \rightarrow j) \in C$. The actual value of $m_{i,j}$ can represent a useful cost metric of $(i \rightarrow j)$ such as, e.g., the number of bytes.

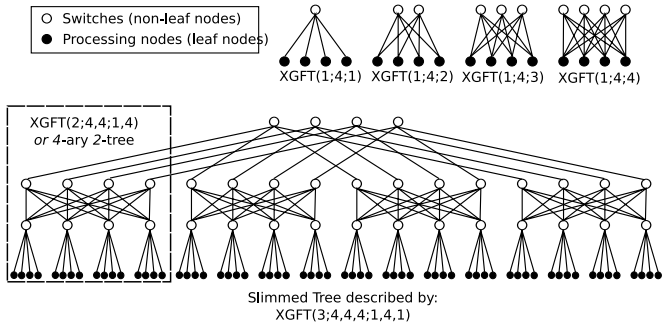


Fig. 1. Several XGFTs

An important kind of communication pattern are permutations: in a permutation every source sends to a distinct destination. The connectivity matrix of a permutation pattern is, naturally, a permutation matrix.

If the network is circuit-switched with end-to-end connections, permutations are the only kind of patterns that can take place at a single point in time. In packet-switched networks, however the situation is quite different. Messages coming from the same source to different destinations or from different sources to the same destination can be present in the network simultaneously, because a message is usually divided into packets that can be interleaved at either of the endpoints. Therefore, at a single point in time, the packets within the network have a more general connectivity matrix than a permutation.

In many HPC applications nodes usually communicate with more than one destination. Application programmers normally choose one of the two following strategies in an effort to maximize the performance of the communication phase: (i) schedule communications such that they form a series of permutations, or (ii) inject all messages in the network at the same time (fragmented into segments) with the objective of taking every possible cycle of progress that would be lost if there was contention in a permutation.

In either case, no general static routing solution exists for k-ary n-trees or slimmed XGFTs that can route all possible permutations without conflict. Oblivious routing algorithms provide a pre-computed set of routes that try to minimize the contention that will apppear in the network. In the following section we will briefly address what contention is, how it affects performance, and what kind of contention can be addressed by a routing scheme. We will then analyze existing routing proposals that have been made to efficiently route as many patterns as possible with minimum contention.

IV. Contention

We can differentiate between two kinds of contention in an application: (i) *endpoint contention*, caused by messages contending for a network adapter because they were produced by or are going to be consumed at the same node[2] and (ii) *routing contention*, caused by messages injected by different nodes and competing for some common switch port. A routing scheme by itself can only address the latter kind.

[2]Assuming that each node has a single network adapter.

Level	# Nodes	Node Labels	# Links	
0	$N^0 = \prod_{j=1}^{h} m_j$	$< M_h, M_{h-1}, \ldots, M_3, M_2, M_1 >$	Down	
			Up	$N^0 \cdot \mathbf{w_1}$
1	$N^1 = \prod_{j=2}^{h} m_j \cdot w_1$	$< M_h, M_{h-1}, \ldots, M_3, M_2, W_1 >$	Down	$= \mathbf{m_1} \cdot N^1$
			Up	$N^1 \cdot \mathbf{w_2}$
2	$N^2 = \prod_{j=3}^{h} m_j \cdot w_1 \cdot w_2$	$< M_h, M_{h-1}, \ldots, M_3, W_2, W_1 >$	Down	$= \mathbf{m_2} \cdot N^2$
			Up	$N^2 \cdot \mathbf{w_3}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
i	$N^i = \prod_{j=i+1}^{h} m_j \cdot \prod_{j=1}^{i} w_j$	$< M_h, \ldots, M_{i+1}, W_i, \cdots, W_1 >$	Down	$\mathbf{m_i} \cdot N^i$
			Up	$N^i \cdot \mathbf{w_{i+1}}$
$i+1$	$N^{i+1} = \prod_{j=i+2}^{h} m_j \cdot \prod_{j=1}^{i+1} w_j$	$< M_h, \ldots, M_{i+2}, W_{i+1}, \cdots, W_1 >$	Down	$= \mathbf{m_{i+1}} \cdot N^{i+1}$
			Up	$N^{i+1} \cdot \mathbf{w_{i+2}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$h-1$	$N^{h-1} = m_h \cdot \prod_{j=1}^{h-1} w_j$	$< M_h, W_{h-1}, \ldots, W_3, W_2, W_1 >$	Down	$\mathbf{m_{h-1}} \cdot N^{h-1}$
			Up	$N^{h-1} \cdot \mathbf{w_h}$
h	$N^h = \prod_{j=1}^{h} w_j$	$< W_h, W_{h-1}, \ldots, W_3, W_2, W_1 >$	Down	$= \mathbf{m_h} \cdot N^h$
			Up	
$M_i \in [0, m_i], W_i \in [0, w_i]$				

TABLE I
THIS TABLE SHOWS THE LABELS ASSIGNED TO THE NODES AND LINKS OF AN XGFT.

We have previously proposed [4] a metric to measure contention by the effective performance loss that a given assignment of routes will cause, rather than by the number of flows assigned to a link. The rationale behind this metric is that flows experiencing endpoint contention can share (part of) their routes without reducing their effective end-to-end bandwidth further, which is the only thing that matters for the global completion time, as messages are being serialized at the edge of the network. Moreover, assigning those flows to the same link (and therefore increasing the number or flows per link) may be beneficial, as more links are left idle for other flows that would otherwise unnecessarily experience network contention (reducing performance further) as opposed to causing more endpoint contention (which is unavoidable and does not depend on the routing scheme).

V. ROUTING IN XGFTS

Finding a minimal deadlock-free path for *connection* $(s \rightarrow d)$ between source node s and destination node d in an XGFT can be done by choosing any of their *Nearest Common Ancestors* (NCAs) [12]. Having selected the NCA, it is very simple to compute the *unique* ascending and descending paths.

Still, assigning the NCAs such that contention is minimized is not easy. Given that there are less NCAs than possible *(source, destination)* pairs, the assignments of NCAs to the communicating pairs of a communication pattern is a combinatorial optimization problem. When the pattern to be optimized is known and is a permutation, and the topology is a k-ary n-tree, a very efficient algorithm exists [15]. When the pattern is known but more general than a permutation, or the topology is not a k-ary n-tree, the problem becomes a hard combinatorial problem.

When the communication pattern is not known, a sensible attempt would be to evenly distribute the NCAs among the different communicating pairs. Several oblivious schemes to do so have been proposed. *Random* routing [16], [17], consists in randomly assigning NCAs to the communicating pairs. It is used as the default mechanism in Myrinet and InfiniBand interconnects. A path $(s \rightarrow d)$ from node s to node d is created by choosing any random NCA between both nodes.

Two other oblivious routing techniques have been proposed independently without a common agreement on their names: we refer to them as *S*ource-*mod-k* routing [1], [10] and *D*estination-*mod-k* routing [6]–[9], [11]. Both techniques employ the same function to select routes. The difference is that the former uses the source node label (See Table I) and the latter the destination label to choose the NCA. S-*mod-k* and D-*mod-k* routings can be concisely described for k-ary n-trees with a simple formula (as the labels form a number in base k): to establish a path $(s \rightarrow d)$ from node s to node d, S-*mod-k* routing chooses parent $\left\lfloor \frac{s}{k^{l-1}} \right\rfloor \bmod k$ at hop l, and D-*mod-k* routing chooses $\left\lfloor \frac{d}{k^{l-1}} \right\rfloor \bmod k$.

Routing in any XGFT is identical to finding paths in k-ary n-trees (see [10]). The routing tables of XGFTs can be filled using straightforward adaptations of the algorithms used in k-ary n-trees. For instance, S-*mod-k* and D-*mod-k* can be adapted by replacing k by w_l in the modulo operation. A route r from s to d that has NCAs at level l_{NCA} is determined by the sequence of local output ports to reach the NCA. Local output ports of switches at level l are numbered from 0 to $w_{l+1} - 1$. Each local output port corresponds to one of the possible parents of the switch reached at level l. A route r is therefore described as: $< r_0, ..., r_l, ..., r_{l_{(s,d)-1}} >$, the path to NCA. The second half of the route to the destination can easily

be reconstructed from the first half by knowing the destination (d) identifier [15].

Routing algorithms for k-ary n-trees can easily be adapted for XGFTs (see [10]). In particular, S-*mod-k* and D-*mod-k* can be easily adapted using the node labels in Table I, which define a variable-radix base numbering for the nodes. To choose the output port at level l, the operation $M_l \bmod w_{l+1}$ is performed. For S-*mod-k* the source's M_l digit is used, and for the D-*mod-k*, the destination's M_l digit.

VI. Workloads and Experimental Methodology

In this Section, the applications chosen as benchmarks are described and the employed methodology and tools are presented.

A. Applications

We have selected two applications that show opposite extreme performance behaviour under commonly used oblivious routings.

1) WRF (Weather Report Forecast) is a numerical weather prediction system designed to serve the atmospheric research community. We include results with 256 processors (WRF-256).
2) The NAS Parallel Benchmarks are a set of pseudo-applications and numerical kernels designed to compare the performance of HPC machines. We present the results for Conjugate Gradient (CG) benchmark with 128 processors for data-set class D: CG.D-128.

B. Tools and Experimental Framework

To study the effect of the routing scheme on network contention, we have used two coupled simulators [18]: Venus and Dimemas. Venus is an event-driven simulator developed at the IBM Zurich Research Laboratory that is able to simulate any generic network topology of nodes, switches, and wires at the flit level. It can simulate any XGFT as well as many other topologies. Dimemas [19] is an MPI simulator driven by a post-mortem trace of a real application execution. The trace contains the MPI calls the application performed, which in turn include the communication pattern as well as the causal relationships between messages. Dimemas reconstructs the temporal behavior according to a parametric bus network model.

For the network model, we have used an input/output buffered switch model, link speed of 2 Gbits/s, flit size of 8 bytes, and segment size of 1 KB with a round-robin interleaving of messages at the network adapter.

We obtained execution traces from runs of the applications selected. Dimemas was fed with the execution trace, relying on Venus to do the detailed network simulation of the communications. We extracted the connectivity matrix M (source-destination pairs) for each communication phase. For each topology under study (instantiations of XGFTs) we fed our routing algorithms with (i) the connectivity matrix, (ii) the topology file, and (iii) the mapping of processes to nodes (sequential). The routes obtained were then supplied, along with the topology and mapping, to the Venus simulator.

We have scaled the reported times against the time employed by a single ideal single-stage crossbar network connecting all the nodes. This network provides the best performance that can be obtained in the absence of network contention. A single-stage network does not need any routing algorithm and does not have any routing contention. We will refer to this network as *Full-Crossbar*.

VII. Experimental and Combinatorial Analysis of S-*mod-k* and D-*mod-k*

The basic conceptual difference between S-*mod-k* and D-*mod-k* in a k-ary n-tree is that while S-*mod-k* concentrates the endpoint contention from the source (every source is assigned a unique path up regardless of the destination), D-*mod-k* works conversely, i.e., every destination is assigned a unique path down regardless of the source. However, S-*mod-k* had been proposed by the earliest works [1], [10], [11] and little attention had been given to it in favor of randomized algorithms [16]. Recently, several works [6]–[9], have analyzed D-*mod-k* and some concluded that it is better than a static random routing [8] or some adaptive routing algorithm [6].

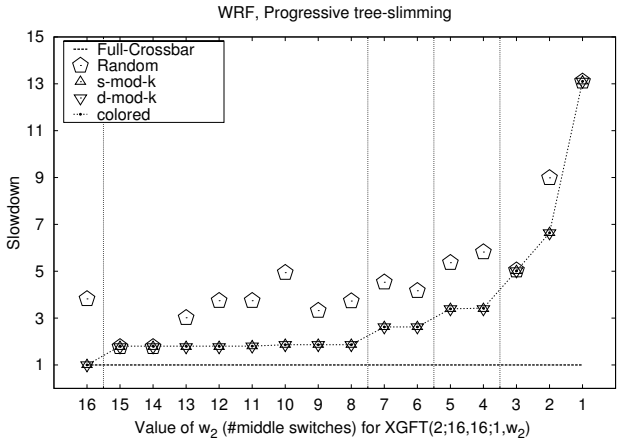
In our experiments (Sec. VII-A and [4]) we saw a negligible difference in performance between S-*mod-k* and D-*mod-k* for the communication patterns used in real applications, prompting the question whether there is an intrinsic difference between the two. We analyzed how many permutations can be routed by either algorithm with a certain level of contention C, and concluded that they are exactly the same. As many applications use more general patterns and the simulations of the cited works have been mainly done with random traffic (which are generally not permutations) it is worth to see whether there are more general patterns routed with a certain level of contention C by S-*mod-k* than by D-*mod-k*. We will show that both algorithms are, in performance terms, for random patterns, equivalent.

When S-*mod-k* and D-*mod-k* are compared against a *Random* routing, *Random* performs better for one of the cases (Section VII-A). One might argue that *Random* better distributes the routes to the roots. A random routing, however, performs worse than either S-*mod-k* and D-*mod-k* for the other application pattern shown in Section VII-A. We studied the distribution of assigned paths per root node of *Random* routing for those patterns and we found that they are better balanced than for S-*mod-k* and D-*mod-k* (Sec. VII-D). That means that balancing of paths to roots, while important, is not the only important factor. *Random* does not assign roots while at the same time concentrating the endpoint contention of the nodes. S-*mod-k* (D-*mod-k*) assign a certain number of sources (destinations) to each NCAs, such that the endpoint contention of those sources (destinations) gets accumulated in the way up (down) to the NCA.

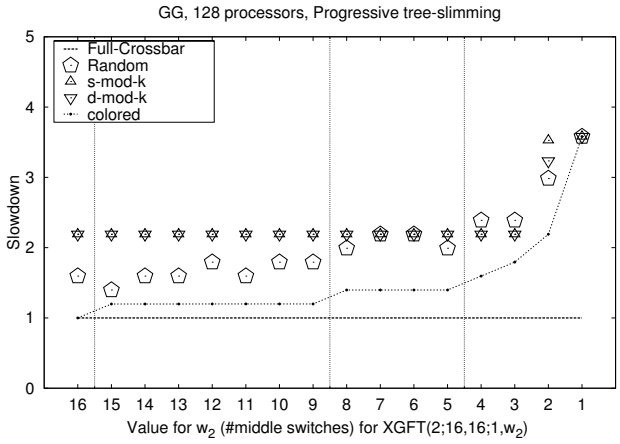
A. Patterns that are not properly routed by S-*mod-k* or D-*mod-k*

Figure 2 shows the slowdown of three oblivious static routings against a pattern-aware routing scheme [4] and the performance that would be obtained with a single-stage network.

The pattern-aware routing scheme (*Colored* [4]), which takes into account the actual communication pattern to optimize the routes, serves as a upper bound for the achievable performance within a network of the same reduced cost as the compared oblivious routing schemes.



(a) WRF: slimmed trees with 32-port switches $m_1, m_2 = 16$



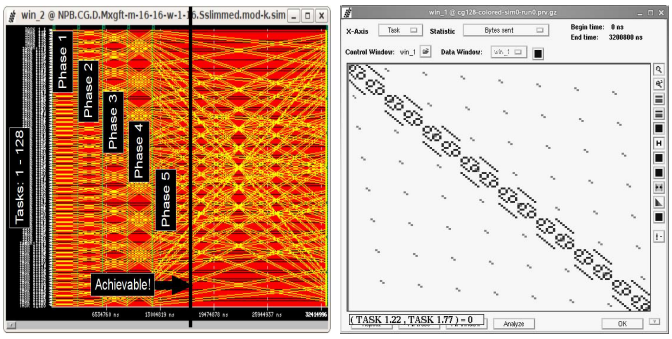
(b) CG.D: slimmed trees with 32-port switches $m_1, m_2 = 16$

Fig. 2. Routing schemes in slimmed-versions of 16-ary 2-trees for two different applications: (a) WRF-256 and (b) CG.D-128. The x axis indicates the number of inner switches of the progressively slimmed topologies with parameters (w_2, ...) of a corresponding slimmed XGFT from the complete k-ary n-tree.

From the analysis of Fig. 2 we can draw the following conclusions: For WRF-256 (Fig. 2(a)) *Random* is worse than the oblivious alternatives S-*mod-k* and D-*mod-k*, which achieve the same performance as a pattern-aware routing scheme. The communication pattern of WRF-256 consists of pairwise exchanges in a 16×16 mesh. Every task (T_i) initiates two outstanding communications to nodes $T_{(i\pm16)}$ (except for the first and last 16 tasks, which only send to nodes $T_{(i+16)}$ and ($T_{(i-16)}$) respectively). Therefore, most of the nodes are either sending to or receiving from two different endpoints. Both S-*mod-k* and D-*mod-k* concentrate the endpoint contention to the same NCAs and assign a different NCA for communicating pairs that do not have endpoint contention. The *Random* scheme, however, introduces additional network contention to the endpoint contention, which explains its poorer behaviour.

On the other side, for CG.D-128 (Fig. 2(b)) we see that *Random* improves over S-*mod-k* and D-*mod-k* in almost all cases. In both WRF-256 and CG.D-128, the performance of S-*mod-k* and D-*mod-k* is almost identical[3] as the communication pattern is symmetric. Yet, for CG.D-128 the performance degradation for S-*mod-k* and D-*mod-k* is huge.

CG has a communication pattern (Fig. 3) that consists of five exchanges of equal size, four of which are local to the first-level switch for the radix[4] we have used $m_1 = 16$. Only the fifth phase is non-local, so whatever degradation in performance this application might suffer due to the routing decision exclusively corresponds to the fifth exchange phase.



(a) Execution trace. (b) Communication matrix.

Fig. 3. CG.D-128 traffic pattern.

We performed a detailed analysis of the performance degradation incurred by D-*mod-k* for the non-slimmed case ($w_2 = 16$), i.e., a $k = 16$-ary 2-tree. There is no contention in the first four phases, which are local to the switch. However, the degradation for the fifth phase (all of equal number of bytes, namely, 750 KB), accounts for more than a factor of two. The simulated trace reveals that this last phase takes eight times longer with D-*mod-k* routing (Fig. 3(a)). This is due to the nature of the communication pattern of CG: each processor s inside a switch communicates to a processor

$$d = \frac{s}{2} \cdot 16 + (s \bmod 2). \tag{2}$$

D-*mod-k* routing will choose $r_1 = (d \bmod 16)$ as the first local port going up into the tree. Given (2), r_1 can only be either 0 for the eight sources within a switch, where $s \equiv 0 \pmod 2$, or 1 for the other eight sources, where $s \equiv 1 \pmod 2$.

In the CG.D-128 case, the regular pattern of the application clashes with the regular assignment of links by the D-*mod-k* routing scheme, as the function used to decide the roots (modulo) is congruent with the pattern itself.

However, if we look at figure 2(b), *Random*, performs better, but still far from the optimum reached by a pattern-aware routing. The fifth phase of CG (the only one that goes out of the switch) is a permutation (Section III). Because it is a permutation, there is no endpoint contention at all and all

[3]The small differences are due to the order in the arrival of packets within a message.

[4]The radix is the k parameter of a k-ary n-tree.

performance degradation is due to network contention. For a full k-ary n-tree (XGFT$(2; 16, 16; 1, 16)$) many optimal solutions exist assigning a different NCA to the communicating pairs such that they do not conflict in the lowermost levels (those close to the leaf nodes). For slimmed trees, not only the NCAs have to be assigned to the communicating pairs such that they do not conflict in the lowermost levels. Because of the reduced connectivity, there will be conflicts in the NCAs as well, and these conflicts should be distributed such that no set of communicating pairs suffers more contention than others; completion time is determined the slowest thread.

B. Number of permutations routed by a S-mod-k or D-mod-k with a certain level of contention C

Given a sequence of source nodes (S) and a sequence of destinations (D), such that D is a permutation P of the sequence S and given a routing table (S-*mod-k*/D-*mod-k*), we can compute the contention level C as the maximum network contention (not accounting for endoint contention) at each of the NCAs assigned to the communicating pairs (C_{NCA_i}). The NCAs for S-*mod-k* are chosen by applying a certain function to the sources in S alone. For the complementary D-*mod-k* routing scheme the function used is the same, but it is applied to the destinations D.

For each communication pattern ($S \rightarrow D$) we can compute the inverse permutation, i.e., $D \rightarrow$S, where the destinations are now the sources, and the sources are the destinations. For this permutation, the NCAs that were previously assigned by S-*mod-k* are now the ones assigned by D-*mod-k*, and vice versa. If we now compute the contention level at the NCAs for ($D \rightarrow S$), with the opposite algorithm as for ($S \rightarrow D$), we will obtain the exact same distribution of contention levels per NCAs, as the sources and destinations have changed place and so has the endpoint in which the routing scheme selects its path.

C. Number of general patterns routed by a S-mod-k or D-mod-k with a certain level of contention C

Any general pattern G can be decomposed into a certain set of permutations, $G = \bigcup_i P_i$. Each of the permutations P_i will achieve a certain contention level c_i, and because the effective bandwidth loss is due to the network contention and not the endpoint contention, we only need to focus on the maximum contention level $c_{max} = \max_i c_i$. Under a S-*mod-k* routing, let's say a particular source s_i appears in several permutations P_i. In the union of permutations, that source will not additionally increase network contention over c_{max}, as source s_i will share the way up in all P_i, increasing endpoint contention only. If we compute the inverse permutations $P_i^{inv} = Inverse(P_i)$ and create the general pattern: $G^{inv} = \bigcup_i P_i^{inv}$, the maximum contention level under the complementary routing scheme (in this example, D-*mod-k*) will be $c_{max}^{inv} = \max_i c_i^{inv} = c_{max}$.

As for any general pattern, an inverse can be found, that will show the exact same behaviour under the converse routing policy, we can safely conclude that both algorithms should perform equally under a well randomized set of permutations.

For application's patterns, if the pattern is symmetric, the inverse is itself, so the number of expected conflicts is the same under both routing schemes. If the pattern is not symmetric, one could be better than other. It is not yet clear which of the two would better apply to a non-symmetric pattern. A possible heuristic would be to choose S-*mod-k* for a many-destinations dominated pattern. And D-*mod-k* for a many-source dominated pattern.

D. Distribution of routes per NCA and the effect on performance

It seems obvious that routes should be evenly distributed across the NCAs. For instance, if in an extreme case all routes go through a single NCA, the effective network is a single k-ary tree (rightmost data points of Fig. 2, with $w_2 = 1$).

Figure 4 shows the distribution of routes per NCAs in a XGFT$(2; 16, 16; 16, 10)$ for different algorithms. *Random* assigns roots evenly to NCAs for the whole set of (source,destination) pairs as can be seen in figure 4(b), but fails to achieve a good performance for WRF-256 (Fig. 2(a)) and a close to optimum performance for CG.D-128 (Fig. 2(b)). S-*mod-k* and D-*mod-k* work much better for WRF-256 despite the uneven distribution of routes to NCAs. The uneven distribution of the S-*mod-k* and D-*mod-k* routing schemes is due to the modulo operation going further up to the tree (going down we have multiples of 16 nodes, but going up, we have 9 roots. Routes for nodes with $M_1 = [10 - 15]$ are assigned to the first five roots as well as those with $M_1 = [0 - 5]$.

If we look at the results for XGFT$(2; 16, 16; 16, 10)$ and the distributions of routes to NCAs of the different algorithms in Fig. 4(a), we see a perfect distribution of routes to NCAs by S-*mod-k* and D-*mod-k*. However, the performance results for CG.D-128 (Fig. 2(b)) are the worst. We might conclude that an even assignment of routes to NCAs is irrelevant.

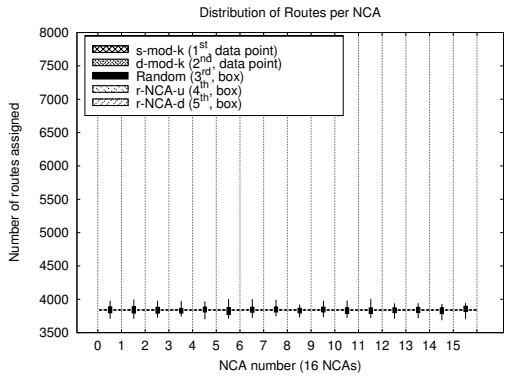
The problem witht the previous example is that we are looking at the *total* number of routes assigned to the NCAs and not the ones effectively used by the communication pattern.

When we know nothing about the communication pattern, it will be better to distribute the NCAs as evenly as possible.

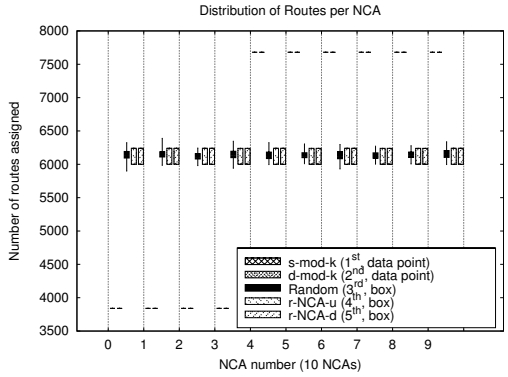
VIII. Proposal: Random NCA Up and Random NCA Down.

The results of the previous section gives us a series of hints to design a class of oblivious algorithms that should behave better for extreme cases such as CG.D-128 and not degrade WRF-256 much.

There are three basic recommendations that lead to better oblivious algorithms: first, concentrate the endpoint contention. This can be seen as assigning "responsabilities" to the root nodes: Each root node will be responsible for a collection of nodes. The responsability taken by the root nodes will be to either concentrate the endpoint contention at the sources (way up) like S-*mod-k*, or to concentrate the endpoint contention at the destinations (way down) like D-*mod-k*. Second, distribute the load among the roots. *Random* does so but does not concentrate endpoint contention, causing unnecessary network



(a) Routes assigned by NCA for an XGFT$(2; 16, 16; 1, 16)$



(b) Routes assigned by NCA for an XGFT$(2; 16, 16; 1, 10)$

Fig. 4. The distribution of the routes assigned per NCA is plotted for several well-known routings and the proposals in this work (H-Rand-D and H-Rand-U) for two XGFTs.

contention. Third, break down the regular dependencies of typical applications' patterns.

The way we assign the responsabilities while at the same time trying to use the self-routing approach is to relabel the sources nodes in a way that applying the modulo of either the source or the destination will randomly distribute the responsabilities of the root nodes, while at the same time assigning the exact same root to each source or destination (depending of whether we are designing a S-*mod-k*-like algorithm or a D-*mod-k*-like one, respectively).

We will first explain the idea behind the relabeling assuming a k-ary n-tree. The relabeling for a k-ary n-tree can be seen as a recursive scramble of the uppermost subtrees (of height $h-1$), and then, independent scrambles of each the lower subtrees of height $h-2$, until we scramble the nodes. Finally the new label given to the node corresponds to the label corresponding to the new position of the node after the scramble. Note that the way the relabeling is done, the topological neighbourhoods of the nodes with respect to their labels are preseved; otherwise the relabeling, and thus the routing would be completely random.

If we want to extend this relabeling to more general XGFTs we have to realize the following: if we give labels based solely on the children per level parameters of an XGFT (the m_i parameter) and then try to use a modulo function to reach the NCA, we will create an unbalance of assignments of routes to NCAs, as seen in Sec. VII-D. If we want to minimize this effect, we have to map the m_i's to w_i's. This leads to repeated labels, but to a better distribution to the NCAs.

Formally, the relabeling can be expressed as follows: given an XGFT$(h; m_1, ..., m_i, ..., m_h; w_1, ..., w_i, ..., w_h)$, and node labels $Node^n =< M_h^n, ..., M_i^n, ..., M_1^n >, n \in [0, \prod_{j=1}^{h} m_j - 1]$, we create new node labels

$$< -, \mathcal{W}^h(M_{h-1}^n), \ldots, \mathcal{W}^2(M_{h-1}^n, \ldots, M_1^n) >$$

where $\mathcal{W}^i(M_{h-1}^n, \ldots, M_i^n)$ are functions that map the interval $[0, \ldots, m_h - 1]$ to the interval $[0, \ldots, w_{h+1}]$.[5]

If $\mathcal{W}^i(M_{h-1}^n, \ldots, M_i^n) = m_i \bmod w_{i+1}$ we will obtain routing strategies D-*mod-k* or S-*mod-k*. We have experimented with random balanced distributions $R_{M_{h-1}^n, \ldots, M_{i+1}^n}(m_i)$.

Once the relabeling is done, we can choose to use the labels either to concentrate the endpoint contention of the sources (in the path up to the NCA) by using the source labels as the guide to reach the NCA (applying S-*mod-k* to the new labels) or to concentrate the contention of the destinations (in the path down from the NCAs) by using the destination labels (applying D-*mod-k* to the new labels).

S-*mod-k* and D-*mod-k* become particular cases of the new class of algorithms. By randomly selecting the NCAs, but still concentrating endpoint contention in the manner of S-*mod-k* or D-*mod-k*, the result over time should avoid pathological cases and improve over a static *Random* strategy.

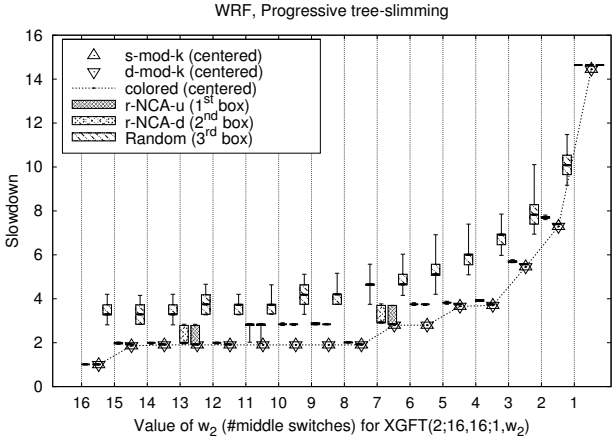
IX. Evaluation

In this section we present the results obtained for non-slimmed and slimmed networks. Figures 5(a) and 5(b) show how diffferent routing algorithms perform when a k-ary n-tree ($w_2 = 16$) is progressively slimmed until it is converted into a k-ary tree ($w_2 = 1$). The X axis records the topology, whereas the Y axis is the slowdown with respect to a Full Crossbar. We use a pattern-aware routing scheme (*Colored* [4]) as a baseline for the best achievable performance for each of the corresponding topologies. We also compare the new routing proposals against S-*mod-k* and D-*mod-k* For the algorithms *Random NCA Up* and *Random NCA Down* we use functions $\mathcal{W}(\ldots)$ that perform a uniform random mapping at each subtree. We use boxplots in the graphs that show the median (as a thick line within the box), and the 25 and 75 percentiles (bottom and top lines of the box), along with the minimum and maximum as whiskerbars. Every box plot is computed from 40 to 60 samples of each algorithm using a different seed.

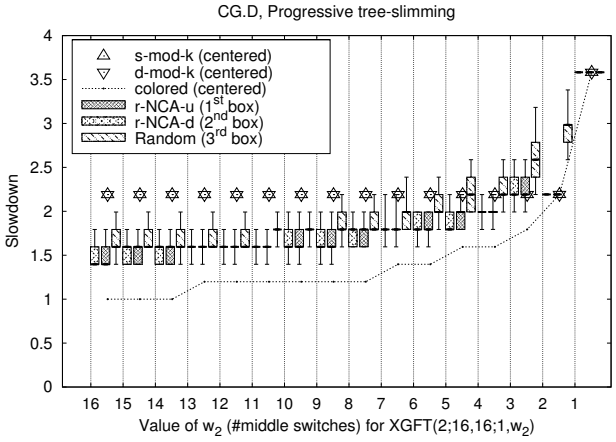
We can see that both for WRF-256 and CG.D-128, the Random NCA Up and Random NCA Down strategies perform statistically better than *Random*. For WRF-256, The performance is always bettern than *Random* and most of the times it is close to the performance of S-*mod-k*, D-*mod-k* or *Colored* for the majority of cases. For the case of CG.D-128, Random NCA Up and Random NCA Down perform statistically better than random for all the cases, and avoids the pathological

[5]The new value for M_h^n (represented as a dash) is irrelevant for the routing scheme. The $\mathcal{W}^i()$ function to map M_1^n is $\mathcal{W}^2()$ because the XGFTs we are considering have $w_1 = 1$.

behaviour of S-*mod-k* and D-*mod-k*. Still, there is a gap to reach the performance of a pattern-aware algorithm such as *Colored*.



(a) WRF: 256 processors



(b) CG.D: 128 processors

Fig. 5. Oblivious Routing Schemes

X. Conclusions and Future Work

In this work we analyzed three widespread oblivious routing algorithms: *Random*, S-*mod-k*, and D-*mod-k*. We identified the reasons that make these algorithm perform well and analyzed two extreme cases. From this analysis we drew a series of conclusions that we used to extend S-*mod-k* and D-*mod-k* to the whole family of XGFTs, and do it as efficiently as possible, while at the same time avoiding pathological cases.

We plan to further improve these algorithms to reduce the gap between their performance in the worst cases and the optimum that is achievable for a specific topology.

XI. Acknowledgements

This work has been partially supported by the Ministry of Science and Technology of Spain under contracts TIN-2007-60625, TIN2007-68023-C02-01, the CONSOLIDER Project CSD2007-00050, the BSC-IBM MareIncognito research agreement and the HiPEAC European Network of Excelence. Thanks also to Mikel Lujan and Rizos Sakellariou (University of Manchester) for their effort in trying to find oblivious solutions for the pathological cases based on the S-*mod-k* and D-*mod-k* algorithms. Their tenacity motivated the current work.

References

[1] C. E. Leiserson *et al.*, "The network architecture of the Connection Machine CM-5," in *Proc. of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, San Diego, CA, USA, Jun. 1992, pp. 272–285.

[2] S. Kamil, J. Shalf, L. Oliker, and D. Skinner, "Understanding ultra-scale application communication requirements," *Proc. Workload Characterization Symposium*, pp. 178–187, Oct. 2005.

[3] N. Desai, P. Balaji, P. Sadayappan, and M. Islam, "Are nonblocking networks really needed for high-end-computing workloads?" in *Proc. 2008 IEEE International Conference on Cluster Computing*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 152–159.

[4] G. Rodriguez, R. Beivide, C. Minkenberg, J. Labarta, and M. Valero, "Exploring pattern-aware routing in generalized fat tree networks," in *ICS '09: Proceedings of the 23rd international conference on Supercomputing*. New York, NY, USA: ACM, 2009, pp. 276–285.

[5] F. Petrini and M. Vanneschi, "k -ary n -trees: High performance networks for massively parallel architectures," *IPPS*, vol. 00, p. 87, 1997.

[6] C. Gomez, F. Gilabert, M. Gomez, P. Lopez, and J. Duato, "Deterministic versus adaptive routing in fat-trees," *Proc. of the* 21^{st} *Parallel and Distributed Processing Symposium, 2007*, pp. 1–8, Mar. 2007.

[7] X.-Y. Lin, Y.-C. Chung, and T.-Y. Huang, "A multiple LID routing scheme for fat-tree-based InfiniBand networks," *Proc. of the* 18^{th} *International Parallel and Distributed Processing Symposium*, pp. 11–, 2004.

[8] G. Johnson, D. J. Kerbyson, and M. Lang, "Optimization of InfiniBand for Scientific Applications." Miami, FL, USA: IEEE, Apr. 2008, pp. 1–8.

[9] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized infiniband fat-tree routing for shift all-to-all communication patterns," 2007.

[10] S. R. Öhring, M. Ibel, S. K. Das, and M. J. Kumar, "On generalized fat trees," in *Proceedings of the* 9^{th} *International Parallel Processing Symposium*. Washington, DC, USA: IEEE Computer Society, 1995, p. 37.

[11] H. Kariniemi, "On-line reconfigurable extended generalized fat tree network-on-chip for multiprocessor system-on-chip circuits," Ph.D. dissertation, Tampere University of Technology, 2006.

[12] F. Petrini and M. Vanneschi, "A comparison of wormhole-routed interconnection networks," in *Proc. Third International Conference on Computer Science and Informatics*, Research Triangle Park, NC, USA, Mar. 1997.

[13] A. Jajszczyk, "Nonblocking, repackable, and rearrangeable Clos networks: fifty years of the theory evolution," *Communications Magazine, IEEE*, vol. 41, no. 10, pp. 28–33, Oct. 2003.

[14] J. Navaridas, J. Miguel-Alonso, F. J. Ridruejo, and W. Denzel, "Reducing complexity in tree-like computer interconnection networks," UPV/EHU, Tech. Rep. EHU-KAT-IK-06-07, 2007.

[15] Z. Ding, R. R. Hoare, A. K. Jones, and R. Melhem, "Level-wise scheduling algorithm for fat tree interconnection networks," in *Proc. 2006 ACM/IEEE Conference on Supercomputing*. New York, NY, USA: ACM, 2006, p. 96.

[16] R. I. Greenberg and C. E. Leiserson, "Randomized routing on fat-trees," in *Proc. of the* 26^{th} *Annual Symposium on the Foundations of Computer Science*, 1985, pp. 241–249.

[17] J. Flich, M. P. Malumbres, P. López, and J. Duato, "Improving routing performance in Myrinet networks," in *Proc. of the* 14^{th} *International Parallel and Distributed Processing Symposium*. Los Alamitos, CA, USA: IEEE Computer Society, 2000, pp. 27–32.

[18] C. Minkenberg and G. Rodriguez Herrera, "Trace-driven co-simulation of high-performance computing systems using OMNeT++," 2009, proc. 2^{nd} International Workshop on OMNeT++, held in conjuction with the Second International Conference on Simulation Tools and Techniques (SIMUTools'09).

[19] J. Labarta, S. Girona, V. Pillet, T. Cortes, and L. Gregoris, "DiP: A parallel program development environment," in *Proc. of the Second International Euro-Par Conference on Parallel Processing*, vol. II. London, UK: Springer-Verlag, 1996, pp. 665–674.